(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 4 443 321 A2**

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
09.10.2024 Bulletin 2024/41

(21) Application number: **24195751.3**

(22) Date of filing: **01.07.2022**

(51) International Patent Classification (IPC):
G06F 21/62 *(2013.01)*

(52) Cooperative Patent Classification (CPC):
G06F 21/6245; G06F 21/6218; G06F 21/6227

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR

(30) Priority: **02.07.2021 US 202163218120 P**

(62) Document number(s) of the earlier application(s) in accordance with Art. 76 EPC:
22754599.3 / 4 185 978

(71) Applicant: **Google LLC**
Mountain View, CA 94043 (US)

(72) Inventors:
- **FOX-EPSTEIN, Eli Simon**
  Mountain View, 94043 (US)
- **YEO, Kevin Wei Li**
  Mountain View, 94043 (US)

(74) Representative: **Crew, Alexander Edward Ross et al**
Kilburn & Strode LLP
Lacon London
84 Theobalds Road
London WC1X 8NL (GB)

Remarks:
This application was filed on 21.08.2024 as a divisional application to the application mentioned under INID code 62.

(54) ENCRYPTED INFORMATION RETRIEVAL

(57) Methods, systems, and computer readable medium facilitating encrypted information retrieval. Methods can include receiving a batch of queries that includes queries to special buckets in each database shard. Query results responsive to the batch of queries are transmitted to the client device. The query results includes server-encrypted secret shares obtained from the special buckets. Client-encrypted versions of the secret shares are received. A full set of server-encrypted secret shares is transmitted to the client device, which is encrypted by the client device to create a full set of client-server-encrypted secret shares. The client device is classified based on how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device.

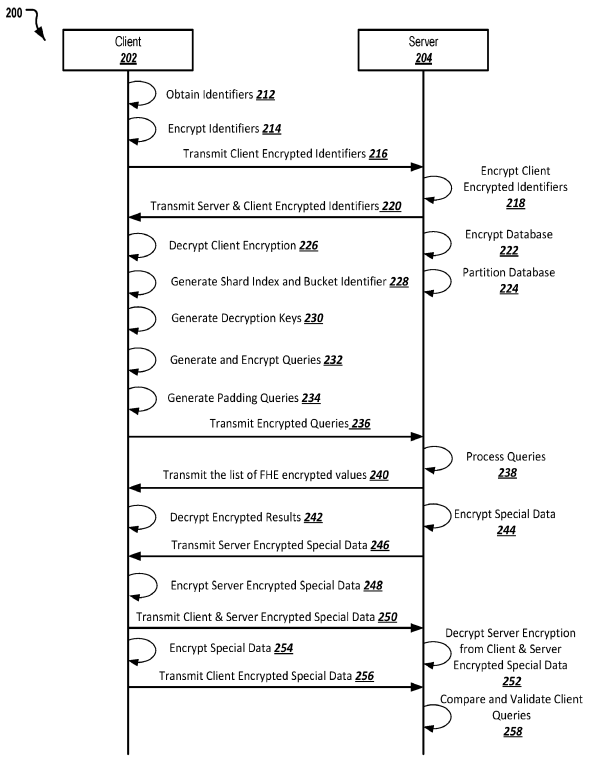

FIG. 2

EP 4 443 321 A2

Description

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit under 35 U.S. C. § 119(e) of U.S. Patent Application No. 63/218,120, entitled "ENCRYPTED INFORMATION RETRIEVAL," filed July 2, 2021. The disclosure of the foregoing application is incorporated herein by reference in its entirety for all purposes.

BACKGROUND

[0002] This specification relates to data processing and information retrieval.

[0003] User devices and content platforms such as content distributors can query content providers in order to retrieve information stored by the content providers. However, there can be situations when it is not in the interest of the content platforms to reveal any details to the content providers about what information is being queried. In other situations, it may not be in the interest of the content providers to reveal any details to the content platforms about other information that is stored on the computing systems of the content providers.

SUMMARY

[0004] In general, one innovative aspect of the subject matter described in this specification can be embodied in methods that include the actions of receiving, at a server device and from a client device, a batch of queries that includes queries to special buckets in each database shard, among multiple database shards, being queried by the batch of queries, wherein the special buckets include server-encrypted secret shares generated by the server; generating, by the server device, a set of query results responsive to the batch of queries, wherein the set of query results includes the server-encrypted secret shares obtained from the special buckets queried by the batch of queries; transmitting, by the server device and to the client device, the set of query results; receiving, at the server device and from the client device, client-encrypted secret shares, wherein the client-encrypted secret shares are client encrypted versions of the secret shares that were included in the set of query results transmitted to the client device; transmitting, by the server device and to the client device, a full set of server-encrypted secret shares, wherein the full set of server-encrypted secret shares includes more server-encrypted secret shares than the set of query results; receiving, at the server device and from the client device, a full set of client-server-encrypted secret shares, wherein the full set of client-server-encrypted secret shares are client encrypted versions of the full set of server-encrypted secret shares that were transmitted to the client device; determining, by the server device, how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device; and classifying, by the server device, the client device based on how many of the secret share are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device.

[0005] Other implementations of this aspect include corresponding apparatus, systems, and computer programs, configured to perform the aspects of the methods, encoded on computer storage devices. These and other implementations can each optionally include one or more of the following features.

[0006] Methods can include removing, by the server device, the server decryption from the full set of client-server-encrypted secret shares received from the client device to obtain a full set of client-encrypted secret shares. Determining how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device can include comparing the client-encrypted secret shares received from the client device to the full set of client-encrypted secret shares obtained by removing the server decryption from the full set of client-server-encrypted secret shares.

[0007] Classifying the client device can include determining that the client device is malicious based on the comparison indicating that fewer than a required number of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-encrypted secret shares obtained by removing the server decryption from the full set of client-server-encrypted secret shares.

[0008] Methods can include receiving, from the client device, a set of client-encrypted entity identifiers; encrypting, by the server, the set of client-encrypted entity identifiers to create a set of sever-client-encrypted identifiers; and transmitting, by the server, the set of server-client-encrypted identifiers to the client device.

[0009] Methods can include generating a partitioned database in which a database is partitioned into the multiple database shards each having a shard identifier that logically distinguishes each database shard from other database shards, and database entries in each database shard are partitioned into buckets having a bucket identifier that logically distinguishes each bucket in the shard from other buckets in the shard.

[0010] Methods can include adding a special bucket to each shard; including, in each special bucket, special data that

is known to the server device, but not the client device; and after each query to a given shard, updating the special data in the special bucket of the given shard to maintain privacy of the information contained in the special bucket of the given shard.

[0011] Methods can include generating, by the client device, a set of queries using the set of server-client-encrypted identifiers; generating, by the client device, a set of decryption keys using the set of server-client-encrypted identifiers; encrypting, by the client device, the set of queries to create the batch of client-encrypted queries.

[0012] The subject matter described in this specification can be implemented in particular embodiments so as to realize one or more of the following advantages. The techniques and methods described in this specification describe techniques for retrieving data from databases while preserving both client and server privacy. This allows a client to query a server without revealing any details to the server about the data that is being queried. Simultaneously, when the client is querying the server does not reveal any details regarding contents of the database that are not queried by the client. In contrast existing techniques of querying a server generally includes encrypting the entire server database and providing the encrypted database to the client for querying. This approach requires significantly more computational resources since the size of the database is generally large. Other approaches of querying the server includes providing an index of the database to the client and receiving a selection of indexes from the client that does not allow for server and client privacy. Furthermore, the subject matter of this application can ensure the client devices are adhering to the agreed upon protocol, for example, by ensuring that the client device is submitting the appropriate number of padding queries, which helps ensure that privacy of the queried data is maintained.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013]

FIG. 1 is a block diagram of an example environment in which content is distributed and presented to a user device.
FIG. 2 is a swim lane diagram of an example process of retrieving content by a client from a server.
FIG. 3 is a flow diagram of an example process of partitioning the server database.
FIG. 4 is a flow diagram of an example process of generating a query from server encrypted identifiers.
FIG. 5 is a flow diagram of an example process of processing queries by a server.
FIG. 6 is a block diagram of an example computer system.

DETAILED DESCRIPTION

[0014] This specification relates to data processing and information retrieval. In particular, the techniques and methods described in this specification describe techniques for retrieving data from databases while preserving both client and server privacy. For example, if the client queries a server database, the client does not reveal any details to the server about the data that is being queried (also referred to as client query privacy). Simultaneously the server does not reveal to the client any details regarding contents of the database that are not queried by the client (also referred to as server database privacy). These techniques enable batch processing of queries to provide for a more efficient information retrieval system that also protects user privacy. For example, user privacy is protected by ensuring that a server being queried cannot learn information about the users that a client is querying the server about, and also prevents the client from learning other information about the users that may be stored by the server.

[0015] FIG. 1 is a block diagram of an example environment 100 in which content is distributed and presented to a user device. The example environment 100 includes a network 102, such as a local area network (LAN), a wide area network (WAN), the Internet, or a combination thereof. The network 102 connects content platforms 106, and content providers 110. The example environment 100 may include many different content providers 110, content platforms 106, and user devices 104.

[0016] A user device 104 is an electronic device that is capable of requesting and receiving content over the network 102. Example user devices 104 include personal computers, mobile communication devices, digital assistant devices, and other devices that can send and receive data over the network 102. A user device 104 typically includes an operating system 112 that is primarily responsible for managing the device hardware and software resources such as applications. The user device 104 also includes a device storage 120 to store data temporarily or permanently based on the particular implementation, application, and use case. A user device 104 typically includes user applications 116 and 117, such as a web browser or an email client, to facilitate the sending and receiving of data over the network 102, but native applications executed by the user device 104 can also facilitate the sending and receiving of content over the network 102. Examples of content presented at a user device 104 include webpages, word processing documents, portable document format (PDF) documents, images, videos, and search results pages and digital ads.

[0017] A content platform 106 is a computing platform that enables distribution of content. Example content platforms 106 include search engines, social media platforms, news platforms, data aggregator platforms, or other content sharing

platforms. Each content platform 106 may be operated by a content platform service provider. The content platform 106 may present content provided by one or more content providers 110. In the above example, the news platform may present content created by different authors and provided by one or more content providers 110. As another example, the content platform 106 may be a data aggregator platform that does not publish any of its own content, but aggregates and presents news articles provided by different news websites (i.e., content providers 110).

[0018] In some implementations, the content platform 106 can distribute digital content to one or more users. For example, the content platform 106 can let one or more users subscribe to and/or register with the content platform 106. In response, the content platform 106 can retrieve digital content from the content providers 110 and provide the retrieved content to the user devices 104 of users. The content provider 110 includes a data storage device (also referred to as a database) that stores digital content in the form of key-value pairs. For example, the database of the content provider 110 can include multiple keys and for each key, a corresponding value that is retrieved by the content platform 106.

[0019] In some implementations, the content platform 106 can assign an identifier to each user so that the content platform can distinguish between the users. In some implementations, the content platform 106 can use information provided by the one or more users and/or the user devices as the unique identifier. For example, the content platform 106 can use, as the unique identifier, an electronic mail identifier (email id) provided by a user, a cell phone number provided by a user, or a media access control (MAC) address of the user device 104 of the user. In some implementations, the content platform 106 can assign an identifier to a group of two or more users based on the characteristics of the users in the group of users. For example, the content platform 106 can assign to a group of users a common identifier based on similar interests in the context of digital content accessed by the users. In another example, users can be assigned to a group and can be assigned a common identifier based on the subscriptions with the digital content. In some implementations, the content platform 106 can assign multiple identifiers to a single user. For example, the content platform 106 can assign both email id and a cell phone number as an identifier for a user.

[0020] To retrieve digital content from the content providers 110, the content platform 106 and the content provider 110 implements an information retrieval technique that ensures data privacy in a way that the content platform 106 does not reveal any details to the content providers 110 about what information is being queried. The retrieval technique further ensures that the content providers 110 does not reveal any details to the content platforms 106 about other information that is stored on the computing systems of the content providers 110. While this specification refers to content providers 110 and content platforms 106, the information retrieval techniques discussed herein can be used by any two systems that want to exchange information in a privacy preserving manner. The information retrieval techniques are further explained with reference to FIG. 2, and entities that are requesting information from a database are referred to as clients and the entities that maintain the database of information, and return information stored in the database are referred to as servers.

[0021] FIG. 2 is a swim lane diagram of an example process 200 of retrieving content by a client from a server. Operations of the process 200 can be implemented, for example, by the client 202 and the server 204. Operations of the process 200 can also be implemented as instructions stored on one or more computer readable media, which may be non-transitory, and execution of the instructions by one or more data processing apparatus can cause the one or more data processing apparatus to perform the operations of the process 200.

[0022] The server implements a database (also referred to as server database) that stores digital content in the form of a mapping between key and value (referred to as a key-value pair). A key can be an identifier and/or a pointer for a value that is the digital content that is being queried by the client. A database can include multiple keys and for each key a respective value corresponding to the key. For example, the server can include, in the database, multiple keys where each key can uniquely identify one or more users of the clients for which the client is retrieving content from the server. In another example, the server can include, in the database, keys that do not necessarily identify users. In such implementations, the value of the corresponding key is data that the server allows clients to query based on other factors such as client permissions or user permissions granted to the client or the user respectively.

[0023] In some implementations, the key of the key-value pair of the server database is associated with the identifier that was assigned to the users by the clients. While querying the server, the client provides the server with a query that includes the identifiers in a way that the details of the identifiers are not disclosed to the server. As explained further in the document, the server can select content (e.g., data of any kind) from the server database based on the identifier even though the identifiers are masked from the server.

[0024] The client 202 obtains unique identifiers (212). For example, the client 202 can provide digital content to one or more users. To uniquely identify the one or more users, the client 202 can assign an identifier to each of the one or more users. In some implementations, the client 202 can use information provided by the one or more users and/or the user devices 104 as the identifiers for the one or more users and/or user devices 104. For example, the client 202 can use, as a unique identifier, an electronic mail identifier (email-id or email address), cell phone number, or media access control (MAC) address of the user devices 104. The client can be any computing system that requests information from another system (e.g., a server system) that stores information or maintains a database.

[0025] The client 202 device encrypts the identifiers (214). To prevent the server 204 from accessing the identifiers

of the users in plaintext, the client 202 encrypts the identifiers using a deterministic and commutative encryption technique to generate an encrypted form of the identifiers (referred to as "client encrypted identifiers"). In general, a commutative encryption is a kind of an encryption that enables a plaintext to be encrypted more than once using different entity's encryption keys. In this system, decryption is not required before the encryption/re-encryption processes. Moreover, the resulting ciphertext (also referred to as encrypted text) can be decrypted by the designated decryption techniques without considering the order of the encryption keys used in the encryption/re-encryption processes. In other words, the order of keys used in encryption and in decryption do not affect the computational result, and allow one encrypting party (e.g., a client 202) to remove their encryption even after another party (e.g., a server) has applied further encryption to data that was encrypted by the first encrypting party.

[0026] The client 202 transmits the client encrypted identifiers to the server 204 (216). For example, after encrypting the identifiers, the client 202 transmits the client encrypted identifiers to the server 204 over the network 102.

[0027] The server 204 encrypts the client encrypted identifiers (218). In some implementations, after receiving the client encrypted identifiers from the client 202, the server 204 re-encrypts (e.g., further encrypts) the client encrypted identifiers using a deterministic and commutative encryption technique to generate an encrypted form of the client encrypted identifiers (referred to as "server & client encrypted identifiers"). In other words, the server 204 adds another layer of encryption on the client encrypted identifiers. Note that the server 204 does not have access to the identifiers in plaintext because the identifiers were already encrypted by the client 202, and the server 204 does not have the decryption key.

[0028] The server 204 transmits the server & client encrypted identifiers back to the client 202 (220). For example, after generating the server & client encrypted identifiers, the server 204 transmits the server & client encrypted identifiers of the one or more users to the client 202 over the network 102.

[0029] The server 204 encrypts the database (222). The server 204 can encrypt the database at any time prior to performing homomorphic operations on the database. The server 204 can also encrypt the database prior to receipt of one or more queries constructed by the client 202 after the client 202 receives the server & client encrypted identifiers. For example, as the server 204 is building the database, or adding information to the database, the server can encrypt the data being stored in the database before homomorphic encryption is applied to the data. In some implementations, the server 204 encrypts the server database using an encryption technique such as an Advanced Encryption Standard (AES). For example, the server 204 encrypts the value of each key-value pair of the server database using the AES encryption technique based on an AES-key generated using the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) and the corresponding key of the key-value pair. Each key of the key-value pair of the server database is further replaced by an integer (referred to as a record key) that is generated using a hash function (e.g., SHA256) that maps the key to an integer within the range [0, n) where n is a tunable parameter known to both the server 204 and the client 202. This results in a record key-AES encrypted value pair in place of each of the key-value pair in the database.

[0030] The hash function can further utilize cryptographic salt added to each key of the key-value pair before hashing. In some implementations, the cryptographic salt is also known to the client 202 that can be used by the client 202 while encrypting the queries.

[0031] The server 204 partitions the database (224). In some implementations, the server partitions the database into shards and each shard into buckets. This is further explained with reference to FIG. 3.

[0032] FIG. 3 is a flow diagram of an example process 300 of partitioning the database into shards and buckets. Operations of the process 300 can be implemented, for example, by the server 204 that includes any entity that implements a database that stores retrievable data. Operations of the process 300 can also be implemented as instructions stored on one or more computer readable media, which may be non-transitory, and execution of the instructions by one or more data processing apparatus can cause the one or more data processing apparatus to perform the operations of the process 300.

[0033] The server 204 partitions the database into P shards (302). For example, by deriving shard indexes from record keys by using the same technique used by the client to derive shard indexes. Each shard may include multiple record key-AES encrypted value pairs.

[0034] The server 204 partitions each shard into buckets (304). In some implementations, the server 204 partitions each shard into even smaller partitions called buckets by deriving a bucket-id for each record key in a shard using the same technique that the client used. For example, shard index and the bucket-id can be computed as the remainder and the integer quotient respectively when the converted number is divided by P. Observe that this yields at most n/P buckets per shard.

[0035] After partitioning each shard into multiple buckets, the AES encrypted values of the record key-AES encrypted value pairs are stored inside each bucket such that the record key of the record key-AES encrypted value pairs indexes both the shard and the bucket inside the shard. It should be noted that the server 204 uses the same methodology as the client 202 to derive a shard number and bucket-id from each key of the key-value pair in the server database.

[0036] The server 204 adds a special bucket to each shard (306). In some implementations, the server adds an additional bucket (referred to as a special bucket) in each shard. Each special bucket of a shard contains data (referred

to as special data) that is known only to the server. For example, if a server database has 10 shards, then each of the 10 shards will include a special bucket and each special bucket will have a unique special data. The special data can be used, for example, to ensure that the client queried an agreed upon number of the special buckets with queries, as discussed in more detail below. In some implementations, the server 204 can update the special data of each special bucket after executing a query from the client 202 so as to maintain privacy regarding the contents of the bucket thereby preventing even a remote possibility of the client 202 or any entity over the network 102 to use that information to sabotage the validation of the client queries described later. In some implementations, the server can add more than one special bucket in each shard.

[0037] In some implementations, the server 204 places the special bucket in a position that is known to the client 202. In other words, the bucket-id of the special bucket is either predefined by the client 202 and the server 204 or is provided to the client 202 by the server 204 at some point during interactions between the client and server. For example, the server 204 can alter the bucket-id at different time-stamps and notify the bucket-id of the special bucket to the client 202 using secure cryptographic protocols. In some implementations, the server 204 can have a different bucket-id for the special bucket for different shards. In such implementations, the server 204 can notify the bucket-id of the special bucket for each shard of the server database.

[0038] The server 204 combines and serializes the encrypted values (308). At this stage, the special bucket included in step 306 is processed in a similar manner as a regular bucket, although, in some implementations, the contents of the special bucket change on a per-query basis. The server 204 concatenates the AES encrypted values of each non-special bucket or the special data of the special buckets into a bytestring. For example, if a particular bucket includes 3 AES encrypted values, the 3 AES encrypted values are concatenated one after the other to generate a bytestring. The server 204 identifies the offset values (index location of the bytestring) of the AES encrypted values and encrypts the offset values using an encryption technique such as AES based on the corresponding record keys. For example, if the bytestring includes 3 AES encrypted values of uniform length, the server encrypts the offset values of each of the 3 AES encrypted values in the bytestring using AES based on the respective record key of the record key-AES encrypted value pair. After generating the encrypted offset values, the server 203 prepends the encrypted offset values to the bytestring. The server 204 further prepends the number of AES encrypted values to the bytestring. In this example, the server 204 prepends the value 3 to the bytestring. The server 204 further splits the bytestring into chunks of c bytes each where c is an integer known in advance to both the client and the server. The c-byte chunks can be further indexed based on their relative position in the bytestring. For example, if c=1, a bucket B can be represented as B = ["p","q","r","s"] where "pqrs" can be the bytestring that is split into c-byte chunks "p", "q", "r" and "s" having indices 1-4 in the bucket respectively. In another example, if c=2, a bucket B can be represented as B = ["pq","rs","tu","v"] where "pqrstuv" can be the bytestring that is split into c-byte chunks "pq", "rs", "tu" and "v" having indices 1-4 in the bucket respectively.

[0039] Now returning to FIG. 2, the client 202 removes the prior client encryption from the server & client encrypted identifiers (226). In some implementations, after receiving the server & client encrypted identifiers, the client 202 uses techniques to decrypt (or remove) the encryption that was performed by the client 202 in step 214 to generate "server encrypted identifiers" for each of the one or more users. Note that the client 202 is able to remove client encryption since the encryption techniques are commutative in nature. Also note that the server encrypted identifiers that are generated after removing the client encryption are identifiers encrypted by the server using the commutative and deterministic encryption technique. In other words, after the client 202 removes the original encryption that was applied to the identifiers, the identifiers remain encrypted by the server, and are then only server encrypted versions of the client identifiers, which are used by the client 202 to generate queries that will be submitted to the server 204 to request information corresponding to the identifiers.

[0040] In some implementations, steps 214-220 and 226 of the process 200 can be implemented using an oblivious pseudo random function (also referred to as an Oblivious PRF or OPRF). An Oblivious PRF is a protocol between a server holding a key to a pseudo random function (PRF) and a client holding an input. At the end of the server-client interaction, the client learns the output of the OPRF on its input provided by the client and nothing else. The server learns nothing about the client's input or the OPRF output.

[0041] To facilitate creation of the queries, the client 202 generates a shard index and a bucket identifier for each server encrypted identifier (228). In some implementations, the client 202 implements a hashing technique to generate a query that can include a shard index and a bucket identifier (also referred to as a bucket-id). An example hashing technique of generating the query is further explained with reference to FIG. 4.

[0042] FIG. 4 is a flow diagram of an example process 400 of generating a query from the server encrypted identifiers. Operations of the process 400 can be implemented, for example, by the client 202 that includes any entity that implements the techniques described in this document to retrieve content from another entity. Operations of the process 400 can also be implemented as instructions stored on one or more computer readable media which may be non-transitory, and execution of the instructions by one or more data processing apparatus can cause the one or more data processing apparatus to perform the operations of the process 400.

[0043] FIG. 4 explains the process 400 of generating a query using an example identifier 450 (john.smith@exam-

ple.com). After processing the identifier 450 using step 226 of the process 200, the server encrypted identifier 460 of the identifier 450 is represented as SERV_ENC{john.smith@example.com}:=adhf8f2g&34!d0sfgn2 where SERV_ENC is the server encryption of the server & client encrypted identifiers after the client 202 removes the client encryption of step 214 and "adhf8f2g&34!d0sfgn2" is the ciphertext of the identifier 450.

[0044] The client 202 hashes the server encrypted identifiers to generate an unsigned integer (410). In some implementations, the client 202 can implement a hash function that is configured to process the server encrypted identifier 460 to generate an unsigned integer 470. This can be represented as HASH_FN[SERV_ENC{john.smith@example.com}]:= 324423510001001110 where HASH_FN is the hash function implemented by the client 202 and "324423510001001110" is the unsigned integer. The hash function (also referred to as a Cryptographic Hash Function) can be any one-way function which is practically infeasible to invert or reverse and that can be used to map data of arbitrary size to a fixed-size value. Examples of such hash functions can include MD5 message-digest algorithm, Secure Hash Algorithm 1, 2 and 3.

[0045] The client 202 converts the unsigned integer into a converted number that is within a specified range (420). In some implementations, the client 202 can implement a conversion function that is configured to process the unsigned integer 470 to generate a converted number 480 that is within a specified range. Since the hash function and the conversion function are known to both the client 202 and the server 204, the range of the converted number generated using the conversion function is pre-specified. In this example, the converted number 480 is represented as CONV[HASH_FN[SERV_ENC{john.smith@example.com }]] := 324425 where CONV is the conversion function implemented by the client 202. As an example, one way to convert an unsigned integer into a number between 0 and 9999 is to use the remainder of the unsigned integer divided by 10000.

[0046] The client 202 splits the converted number into a shard index and a bucket-id (430). The shard index will be a number between 0 and P-1, where P is the number of shards. The bucket-id will range between 0 and n/P-1, where n is the largest value that the converted number 480 generated in step 420 can take. In some implementations, the client 202 splits the converted number 480 generated in step 420 into two parts such that the first part is the shard index and the second part is the bucket-id. For example, if n is the largest value that the converted number 480 can take, the number of bits required to represent the number n is $log_2\ n$ bits. In such a scenario, if P is a power of 2 where P = 2K, the client 202 can use the first K bits as the shard index and the remaining, $log_2\ n$ - K bits can be used as a bucket-id. In this example, the client 202 splits the converted number 480 into two parts as depicted in FIG. 3 as result 490 such that the shard index is 32 and the bucket-id is 4425. In another implementation, if P represents the number of shards, the shard index can be computed as the remainder when the converted number is divided by P. In this case, the bucket-id can be computed as the integer quotient when the converted number is divided by P. In this case, if P=100 then shard index 25 and bucket-id 3244.

[0047] Returning back to the process 200, the client 202 uses the process 400 to generate a query for each of the server encrypted identifiers of the one or more users. For example, each query that is generated for each identifier will include the shard index and the bucket id created using the server encrypted identifier, as discussed above.

[0048] The client 202 generates decryption keys using each server encrypted identifier (230). In some implementations, the client 202 can generate a decryption key for each of the server encrypted identifiers. For example, the client 202 can implement a HMAC-based Extract-and-Expand Key Derivation Function (HKDF) which is a hash-based message authentication code (HMAC) cryptographic key derivation function (KDF) for expanding a key into one or more cryptographically strong secret keys. For example, the client 202 can use the HKDF to process the server encrypted identifier to generate a decryption key.

[0049] The client 202 generates and encrypts queries (232). In some implementations, the client 202 uses the bucket-id computed using the process 300 to generate an indicator vector of length n/P where the element with index equal to the bucket-id is 1 and the other elements are 0 (recalling that P is the number of shards and n/P is the number of distinct bucket-ids). In some implementations, the indicator vector can be compressed using well-known compression techniques. In some implementations, the client 202 can encrypt the indicator vector corresponding to each of the server encrypted identifiers using a fully homomorphic encryption (FHE) technique to generate a corresponding FHE encrypted bucket vector. In general, homomorphic encryption is a form of encryption that permits users to perform computations on its encrypted data without first decrypting it. These resulting computations are left in an encrypted form which, when decrypted, result in an identical output to that produced had the operations been performed on the unencrypted data. Properties of FHE can include addition, multiplication and absorption. To illustrate, if {x} denotes a FHE of x, then the addition {x} + {y} can yield {x+y} without revealing x, y or x+y. Similarly, the multiplication {x} * {y} can yield {xy} without revealing x, y or xy and the absorption {x} * y can yield {xy} without revealing x. In some implementations, the properties of FHE can include the ability to convert an encrypted vector into a specified number of separate encrypted values (one for each item in the vector) without revealing any details about the items in the vector.

[0050] After encrypting the indicator vector, the client 202 can generate a query that includes a shard index and a corresponding FHE encrypted bucket vector. The query can be denoted as FHE QUERY (database, query) where the database represents a data storage that stores a mapping of key and value i.e., the database stores multiple keys and

for each key a corresponding value.

[0051] The client 202 generates padding queries (234). In general, the number of unique identifiers for which the client 202 is retrieving data from the server database is less than the number of unique identifiers in the server database. In some implementations, in order to mask the true queries and their distribution to the shards generated in step 232 of the process 200 from the server 204, the client 202 can generate padding queries that can be transmitted to the server 204. The client 202 can determine the number of padding queries required, for example, based on information provided by the server 204. For example, the server 204 can specify that a certain number of padding queries (e.g., 99 padding queries or another appropriate number of padding queries) is required to be generated and submitted by the client 202 for each real query being submitted by the client 202. The number of padding queries required to be submitted in various ways. For example, the number of padding queries can be done using theoretical or experimental approaches to pick a fixed number of queries per shard to issue, or even heuristically or arbitrarily selected by an entity (e.g., an administrator) of the server 204. In another example, assume that the server database is partitioned into 5 shards and that the server 204 and client 202 have agreed on 3 queries per shard i.e., the client 202 can transmit a maximum of 3 queries for each shard. Further assume that the client 202 generates 7 real queries that are distributed as 0 real queries for the first shard, 3 real queries for the second shard, 1 real query for the third shard, 2 real queries for the fourth shard and 1 real query for the fifth shard. In this example, the client 202 needs to generate a total of 8 padding queries that can be computed using the following equation

$$\begin{aligned} &Number\ of\ padding\ queries \\ &\quad = (Number\ of\ shards\ *\ Number\ of\ queries\ per\ shard) \\ &\quad - number\ of\ real\ queries \end{aligned}$$

[0052] The number of padding queries generated by the client 202 are distributed as 3 padding queries for the first shard, 0 padding queries for the second shard, 2 padding queries for the third shard, 1 padding queries for the fourth shard and 2 padding queries for the fifth shard.

[0053] In some implementations, the client 202 uses the bucket-id of the special buckets added by the server 204 computed using the process 300 to generate an indicator vector for the special buckets. Similar to the step 232 of the process 200, the indicator vector for the special buckets can be compressed using well-known compression techniques and encrypted using a fully homomorphic encryption (FHE) technique to generate a corresponding FHE encrypted bucket vector for the special buckets.

[0054] In some implementations, each padding query is directed to the special buckets of the shards being queried by the real queries (e.g., queries generated using the identifiers). For example, to query the database in a privacy preserving manner, the client may be required to generate 99 padding queries for each real query. In this example, each of the 99 padding queries will be formed such that they are directed to the special buckets, and will be queries for the special data contained in the special buckets. In some implementations, the special data can be random values or other data. As discussed further below, each padding query will return a different special data stored in one of the special buckets, which can be used by the server to ensure that the client device generated the required number of padding queries (e.g., by comparing the special data returned in response to the queries to the full set of special data known to the server). If the special data in each of the special buckets is different, then if the number of distinct special data provided to the client matches the number of padding queries required to be created by the client device, the server can conclude that the client 202 is not compromised and/or malicious.

[0055] The client 202 transmits the query to the server 204 (236). For example, after generating the queries for each of the server encrypted identifiers and the padding queries, the client 202 transmits the full set of queries to the server 204 over the network 102. In some implementations, multiple queries are sent to the server in a batch, such that the server can process the queries at the same time (e.g., in a batch process).

[0056] The server 204 processes the queries (238). At this stage, the real queries and the padding queries are processed in a similar way. Note that the server 204 has no information to identify the real queries from the padding queries. In some implementations, the server 204 can process the queries using an optimized batch process that reduces the resource consumption required to retrieve content from the database. The optimized process can be implemented in a manner that facilitates concurrent processing by splitting the database into smaller chunks (referred to as shards and identified using shard index), and processing them in parallel on multiple computing systems thereby reducing the computational resources required to retrieve content from the database. This is further explained with reference to FIG. 5.

[0057] FIG. 5 is a flow diagram of an example process 500 of processing queries. Operations of the process 500 can be implemented, for example, by the server 204 that includes any entity that implements a database from where content is being retrieved. Operations of the process 500 can also be implemented as instructions stored on one or more computer readable media which may be non-transitory, and execution of the instructions by one or more data processing apparatus

can cause the one or more data processing apparatus to perform the operations of the process 500.

[0058] For each query, the server 204 identifies the shard using the shard-index of the query (502). As mentioned before with reference to step 232 and 234 of the process 200, each query includes a shard index and a corresponding FHE encrypted bucket vector. After receiving a query, the server 204 identifies a particular shard based on the shard index of the query. For example, if the shard index of the query is 32, the server identifies the 32nd shard based on the shard indexes of the server database. Furthermore, the system can be designed so that each of the shards is selected the exact same number of times to prevent any information about the queries from being gleaned based on the shards queried.

[0059] For each query, the server 204 queries each bucket in the shard and generates a list of FHE encrypted values (504). In some implementations, the server 204 queries each bucket of the particular shard identified by the shard index using the FHE encrypted bucket vector from the query. For example, if there are 32 buckets in a shard that was identified using the shard-index from the query, the server 204 will query each of the 32 buckets.

[0060] There are a variety of ways that buckets can be queries, and any appropriate bucket querying technique can be used. For example, the server 204 performs an oblivious expansion operation on the FHE encrypted bucket vector from the query to obtain an FHE encrypted value for the particular bucket. Then it performs a separate FHE absorption operation between the FHE encrypted value for the particular bucket and each c-byte chunk in the bucket. This can be logically explained with the following non-limiting example.

[0061] Assume that there are 4 buckets in the particular shard. Further, assume that the 1st bucket has the following chunks ["A", "B", "C", "D"]. Similarly the 2nd, the 3rd and the 4th bucket has the following chunks ["E", "F", "G"], ["H"] and ["I", "J", "K"] respectively. Further, assume that the indicator vector is [0, 1, 0, 0]. An absorption operation will generate FHE encrypted values of chunks with index 1 across all four buckets that can be represented as [0, "E", 0, 0]. Similarly FHE encrypted values of chunks with indices 2-4 across all four buckets are [0, "F", 0, 0], [0, "G", 0, 0] and [0, 0, 0, 0] respectively.

[0062] In some implementations, the server 204 can aggregate the values of the FHE encrypted values of the bucket vector and the c-byte chunks using the FHE addition operation across all buckets and generate a list of FHE encrypted values. In other words, all entries in the set of triples described previously with the same query and chunk _index are combined into one by summing the FHE values. For example, the aggregation operation on the values of the FHE encrypted values of the bucket vector and the c-byte chucks with a particular index, for example, index 1 will select the chunk "E" from among all chunks having index 1 across all four buckets of the shard. Similarly the aggregated values of the chunks at the 2nd, the 3rd and the 4th indices are "F","G" and 0 respectively across all buckets of the shard. The server 204 after selecting the chunks from buckets, the server 204 can generate a list of FHE encrypted values and transmits the list to the client 202. For example, the server can generate a list ["E", "F", "G"] of FHE encrypted values that were selected using the absorption operation and transmit the list to the client 202. As mentioned above, the server 204 has no way of identifying the real queries from the padding queries. Therefore, the padding queries are processed in the same way as the real queries, and the server 204 selects the special data from the special buckets for the corresponding shards based on the padding queries. The structure of the padding queries is pre-specified so that querying the shard with the padding query results in the server returning the special data in response to the padding query.

[0063] The process 500 is implemented in a way that multiple queries are processed in parallel on multiple computing systems. In some implementations, a map reduce process can be used that enables all queries to be processed as a batch of queries, which reduces the time required to generate a response to the queries, and saves processing resources relative to processing each query individually. For example, assume that the database is partitioned into n buckets by hashing keys and that the buckets are partitioned into shards based on leading k bits. In this example, the server can partition the queries by shard based on the provided shard index that is submitted with each query. The server can then fan out each query to an FHE value per (existing) bucket in its shard by decompressing the encrypted bucket selector. Within each shard, each bucket is joined with the FHE values from the queries for the bucket. For each bucket: for each pair in the cartesian product of FHE values from queries and chunks of the bucket, perform an FHE absorption. The output of this step is a multimap from (query_id, chunk index) pairs to FHE-encrypted values. The values are aggregated using FHE addition as the aggregator. This has the same output format as the previous step, except that it's not a multimap - each key has exactly one FHE value. A list of encrypted values sorted by chunk_index is aggregated, and the output format is a map from query to a list of FHE-encrypted values. By providing the same number of queries per shard and with appropriate sharding, the computational costs can be reduced by having many shards without revealing any information about the distribution of queries. Now returning back to FIG. 2.

[0064] The server 204 transmits the list of FHE encrypted values to the client 202 (240). The server 204 after generating the list of FHE encrypted values of the bucket vector and the c-byte chunks for each queries, transmits the one or more lists to the client 202 over the network 102.

[0065] The client 202 decrypts the FHE encryption (242). In some implementations, after receiving the lists of FHE encrypted values of the bucket vector and the c-byte chunks for each of the queries transmitted by the client 202, the client 202 can segregate the FHE encrypted values for the padding queries from the real queries because the client

device knows which queries were padding queries. At this point the client can use the decryption keys that were generated in step 230 of the process 200 to decrypt the FHE encryption to obtain the value of the key-value pair that was queried and was originally stored on the server database. While decrypting the FHE encrypted values, the client 202 also obtains the special data that was retrieved from the server database for each of the padding queries transmitted by client 202 in step 236. The special data that was retrieved from the server database for each of the padding queries is all together referred to as a retrieved set of special data.

[0066] The server 204 encrypts the special data of each special bucket (244). In some implementations, the server 202 encrypts the special data of each special bucket using a commutative encryption technique (either deterministic or non-deterministic) to generate a corresponding server encrypted special data. This generates a universal set of server encrypted special data that includes encrypted values of each unique special data from each unique special bucket added by the server 204. The server 204 can encrypt the universal set of server encrypted special data at any point before or after receiving the queries from the client device 202.

[0067] The server 204 transmits the server encrypted special data to the client 202 (246). The server 204 after generating the universal set of server encrypted special data, transmits the universal set to the client 202 over the network 102. The client 202 is unable to decrypt the universal set of server-encrypted special data, but as discussed below, the client 202 can apply further encryption that will allow the server to compare client encrypted versions of the special data that were returned to the client device 202 in the query results returned to the client device 202 responsive to processing the queries with the universal set of server encrypted special data.

[0068] The client 202 encrypts the server encrypted special data to create client server encrypted special data (248). In some implementations, after receiving the universal set of server encrypted special data from the server 204, the client 202 re-encrypts (e.g., further encrypts) the server encrypted special data of the universal set using a deterministic and commutative encryption technique to generate an encrypted form of the server encrypted special data (referred to as "client & server encrypted special data"). This generates a universal set of client & server encrypted special data. Generally, the universal set of client & server encrypted special data will include more special data than the retrieved set of special data that are provided to the client device 202 in response to the queries. More specifically, the universal set of client & server encrypted special data will contain as many special data as the total number of queries that includes both real queries and padding queries.

[0069] The client 202 transmits the universal set of client & server encrypted special data to the server 204 (250). The server 204 receives the universal set of client & server encrypted special data, and as discussed below, processes the universal set of client & server encrypted special data to determine whether the client device 202 used the required number of padding queries (e.g., a specified number of padding queries for each real query).

[0070] The server 204 removes the prior server encryption from the universal set of client & server encrypted special data to obtain a universal set of client encrypted special data (252). After receiving the universal set of client & server encrypted special data, the server 204 uses techniques to decrypt (or remove) the encryption that was performed by the server 204 in step 244. Once the server removes the previously applied server encryption from each of the client & server encrypted special data in the universal set of client & server encrypted special data, the result is the full set of client encrypted universal set of client encrypted special data. The universal set of client encrypted special data includes special data of each unique special bucket that is encrypted using client encryption, but not server encryption. In some implementations, steps 244-252 of the process 200 can be implemented using an Oblivious PRF. It should be noted that steps 244-252 of the process 200 can be performed at any point after the server 204 has added the additional buckets. For example, after step 306 of the process 300.

[0071] The client 202 encrypts the special data retrieved from the server database to generate client-encrypted special data (254). In some implementations, the client 202, after obtaining the retrieved set of special data, encrypts each of the special data in the retrieved set using the same commutative encryption technique as used in step 248 of the process 200 to generate a retrieved set of client encrypted special data. The retrieved set of client encrypted special data does not include any server encryption, and therefore, can be compared to the universal set of client encrypted special data to determine how many matches occur without revealing the underlying special data, because revealing the underlying special data could reveal to the server which queries were padding queries. The number of matches can be used, for example, by the server to ensure that the client device is adhering to the agreed upon protocol without requiring the client device to reveal the information it actually submitted in the queries. For example, as described below, the server device can validate that the client device has submitted the agreed upon minimum number of padding queries based on the number of matches identified. This ensures that the client device is submitting a sufficient number of padding queries, which helps ensure the privacy of the underlying data being queried by client devices.

[0072] The client 202 transmits the client encrypted special data to the server 204 (256). The server 204 receives the retrieved set of client encrypted special data over the network 102, and further processes the client encrypted special data, as discussed below.

[0073] The server 204 compares and validates the client queries (258). In some implementations, after receiving the retrieved set of client encrypted special data, the server 204 checks whether the client encrypted special data is a proper

subset of the universal set of client encrypted special data generated in step 252 of the process 200. In other words, the server 204 checks whether each of the client encrypted special data in the retrieved set is included in the universal set of client encrypted special data. Although the server 204 cannot access the underlying special data in either the universal set of client encrypted special data or the retrieved set of client ncrypted special data generated using the special data provided to the client 202 in the query results, the server 204 can compare the encrypted versions to determine whether the underlying data matches. If there are one or more client encrypted special data that are in the retrieved set and not the universal set, the server 204 can conclude that the client 202 is compromised and/or malicious because it did not submit the required number of padding queries. In some implementations, the server 204 can compute the cardinality of intersection of the retrieved set of client encrypted special data and the full set of client encrypted special data. If the cardinality of the intersection is not equal to the number of padding queries (known to the server), the server 204 can conclude that the client 202 is compromised and/or malicious.

[0074] In some implementations, the server 204 can withhold some information after concluding that the client 202 is compromised and/or malicious to prevent the client 202 from accessing the data that was retrieved from the server database. For example, the server 204 can add another layer of encryption to the values of the key-value pairs of the server database using any known encryption techniques based on an encryption key that is known only to the server 204. If the server 204 concludes that the client 202 was compromised and/or malicious, the server 204 will not transmit the encryption key to the client 202 thereby preventing the client from accessing the retrieved data in cleartext. On the contrary if the server 204 concludes that the client 202 is not compromised and/or malicious, the server 204 will transmit the encryption key to the client 202 thereby allowing the client 202 to access the retrieved data in cleartext. In another example, the server 204 can withhold the cryptographic salt used by the HKDF to derive AES keys.

[0075] In some implementations, the server 204 can validate whether the client 202 is not compromised or malicious by implementing a secret sharing scheme such as a Shamir's Secret Sharing (SSS). In such implementations, the server 204 generates secret shares of a secret data that is known only to the server 204. The server 204 includes a secret share into the special bucket of each shard, such that for each query, the server 204 is placing a secret share into the special bucket for each shard. As noted above, the secret share included in each bucket can be changed on a per-query basis. Similar to how the client retrieved special data corresponding to each of the padding queries in step 242, the client 202 retrieves secret shares corresponding to each of the padding queries. The client 202 tries to reconstruct the secret data using the retrieved secret shares and transmits the secret data to the server 204. The server 204 after receiving the secret data, compares the secret data received from the client 202 to the secret data that was originally known only to the server 204. If the two secret data do not match, the server 204 can conclude that the client 202 is compromised and/or malicious. The parameters of the secret sharing scheme should be chosen in a way that only if the client 202 honestly queries for the special data as specified, the client 202 can then retrieve the encoded secret,

[0076] FIG. 6 is a block diagram of an example computer system 600 that can be used to perform operations described above. The system 600 includes a processor 610, a memory 620, a storage device 630, and an input/output device 640. Each of the components 610, 620, 630, and 640 can be interconnected, for example, using a system bus 650. The processor 610 is capable of processing instructions for execution within the system 600. In some implementations, the processor 610 is a single-threaded processor. In another implementation, the processor 610 is a multi-threaded processor. The processor 610 is capable of processing instructions stored in the memory 620 or on the storage device 630.

[0077] The memory 620 stores information within the system 600. In one implementation, the memory 620 is a computer-readable medium. In some implementations, the memory 620 is a volatile memory unit. In another implementation, the memory 620 is a non-volatile memory unit.

[0078] The storage device 630 is capable of providing mass storage for the system 600. In some implementations, the storage device 630 is a computer-readable medium. In various different implementations, the storage device 630 can include, for example, a hard disk device, an optical disk device, a storage device that is shared over a network by multiple computing devices (e.g., a cloud storage device), or some other large capacity storage device.

[0079] The input/output device 640 provides input/output operations for the system 600. In some implementations, the input/output device 640 can include one or more of a network interface devices, e.g., an Ethernet card, a serial communication device, e.g., and RS-232 port, and/or a wireless interface device, e.g., and 802.11 card. In another implementation, the input/output device can include driver devices configured to receive input data and send output data to external devices 560, e.g., keyboard, printer and display devices. Other implementations, however, can also be used, such as mobile computing devices, mobile communication devices, set-top box television client devices, etc.

[0080] Although an example processing system has been described in FIG. 5, implementations of the subject matter and the functional operations described in this specification can be implemented in other types of digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them.

[0081] Embodiments of the subject matter and the operations described in this specification can be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Embodiments of the subject

matter described in this specification can be implemented as one or more computer programs, i.e., one or more modules of computer program instructions, encoded on computer storage media (or medium) for execution by, or to control the operation of, data processing apparatus. Alternatively, or in addition, the program instructions can be encoded on an artificially-generated propagated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal that is generated to encode information for transmission to suitable receiver apparatus for execution by a data processing apparatus. A computer storage medium can be, or be included in, a computer-readable storage device, a computer-readable storage substrate, a random or serial access memory array or device, or a combination of one or more of them. Moreover, while a computer storage medium is not a propagated signal, a computer storage medium can be a source or destination of computer program instructions encoded in an artificially-generated propagated signal. The computer storage medium can also be, or be included in, one or more separate physical components or media (e.g., multiple CDs, disks, or other storage devices).

[0082] The operations described in this specification can be implemented as operations performed by a data processing apparatus on data stored on one or more computer-readable storage devices or received from other sources.

[0083] The term "data processing apparatus" encompasses all kinds of apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, a system on a chip, or multiple ones, or combinations, of the foregoing. The apparatus can include special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit). The apparatus can also include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, a cross-platform runtime environment, a virtual machine, or a combination of one or more of them. The apparatus and execution environment can realize various different computing model infrastructures, such as web services, distributed computing and grid computing infrastructures.

[0084] A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, declarative or procedural languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, object, or other unit suitable for use in a computing environment. A computer program may, but need not, correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub-programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers that are located at one site or distributed across multiple sites and interconnected by a communication network.

[0085] The processes and logic flows described in this specification can be performed by one or more programmable processors executing one or more computer programs to perform actions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit).

[0086] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors. Generally, a processor will receive instructions and data from a read-only memory or a random access memory or both. The essential elements of a computer are a processor for performing actions in accordance with instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks. However, a computer need not have such devices. Moreover, a computer can be embedded in another device, e.g., a mobile telephone, a personal digital assistant (PDA), a mobile audio or video player, a game console, a Global Positioning System (GPS) receiver, or a portable storage device (e.g., a universal serial bus (USB) flash drive), to name just a few. Devices suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[0087] To provide for interaction with a user, embodiments of the subject matter described in this specification can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input. In addition, a computer can interact with a user by sending documents to and receiving documents from a device that is used by the user; for example, by sending web pages to a web browser on a user's client device in response to requests received from the web browser.

[0088] Embodiments of the subject matter described in this specification can be implemented in a computing system

that includes a back-end component, e.g., as a data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the subject matter described in this specification, or any combination of one or more such back-end, middleware, or front-end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a local area network ("LAN") and a wide area network ("WAN"), an inter-network (e.g., the Internet), and peer-to-peer networks (e.g., ad hoc peer-to-peer networks).

[0089] The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other. In some embodiments, a server transmits data (e.g., an HTML page) to a client device (e.g., for purposes of displaying data to and receiving user input from a user interacting with the client device). Data generated at the client device (e.g., a result of the user interaction) can be received from the client device at the server.

[0090] While this specification contains many specific implementation details, these should not be construed as limitations on the scope of any inventions or of what may be claimed, but rather as descriptions of features specific to particular embodiments of particular inventions. Certain features that are described in this specification in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.

[0091] Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system components in the embodiments described above should not be understood as requiring such separation in all embodiments, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.

[0092] Thus, particular embodiments of the subject matter have been described. Other embodiments are within the scope of the following claims. In some cases, the actions recited in the claims can be performed in a different order and still achieve desirable results. In addition, the processes depicted in the accompanying figures do not necessarily require the particular order shown, or sequential order, to achieve desirable results. In certain implementations, multitasking and parallel processing may be advantageous.

[0093] Further, the present disclosure includes the following clauses:

Clause 1. A method, comprising:

receiving, at a server device and from a client device, a batch of queries that includes queries to special buckets in each database shard, among multiple database shards, being queried by the batch of queries, wherein the special buckets include server-encrypted secret shares generated by the server;

generating, by the server device, a set of query results responsive to the batch of queries, wherein the set of query results includes the server-encrypted secret shares obtained from the special buckets queried by the batch of queries;

transmitting, by the server device and to the client device, the set of query results;

receiving, at the server device and from the client device, client-encrypted secret shares, wherein the client-encrypted secret shares are client encrypted versions of the secret shares that were included in the set of query results transmitted to the client device;

transmitting, by the server device and to the client device, a full set of server-encrypted secret shares, wherein the full set of server-encrypted secret shares includes more server-encrypted secret shares than the set of query results;

receiving, at the server device and from the client device, a full set of client-server-encrypted secret shares, wherein the full set of client-server-encrypted secret shares are client encrypted versions of the full set of server-encrypted secret shares that were transmitted to the client device;

determining, by the server device, how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device; and

classifying, by the server device, the client device based on how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted

secret shares received from the client device.

Clause 2. The method of clause 1, further comprising:
removing, by the server device, the server decryption from the full set of client-server-encrypted secret shares received from the client device to obtain a full set of client-encrypted secret shares, wherein determining how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device comprises comparing the client-encrypted secret shares received from the client device to the full set of client-encrypted secret shares obtained by removing the server decryption from the full set of client-server-encrypted secret shares.

Clause 3. The method of clause 2, wherein classifying the client device comprises determining that the client device is malicious based on the comparison indicating that fewer than a required number of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-encrypted secret shares obtained by removing the server decryption from the full set of client-server-encrypted secret shares.

Clause 4. The method of clause 3, further comprising:

receiving, from the client device, a set of client-encrypted entity identifiers;
encrypting, by the server, the set of client-encrypted entity identifiers to create a set of sever-client-encrypted identifiers;
transmitting, by the server, the set of server-client-encrypted identifiers to the client device.

Clause 5. The method of clause 4, further comprising:
generating a partitioned database in which a database is partitioned into the multiple database shards each having a shard identifier that logically distinguishes each database shard from other database shards, and database entries in each database shard are partitioned into buckets having a bucket identifier that logically distinguishes each bucket in the shard from other buckets in the shard.

Clause 6. The method of clause 5, further comprising:

adding a special bucket to each shard;
including, in each special bucket, special data that is known to the server device, but not the client device; and
after each query to a given shard, updating the special data in the special bucket of the given shard to maintain privacy of the information contained in the special bucket of the given shard.

Clause 7. The method of clause 6, further comprising:

generating, by the client device, a set of queries using the set of server-client-encrypted identifiers;
generating, by the client device, a set of decryption keys using the set of server-client-encrypted identifiers;
encrypting, by the client device, the set of queries to create the batch of client-encrypted queries.

Clause 8. A system comprising:

a database configured to store data; and
a server device configured to process queries using the database and execute instructions that cause the server device to perform operations comprising:

receiving, from a client device, a batch of queries that includes queries to special buckets in each database shard, among multiple database shards, being queried by the batch of queries, wherein the special buckets include server-encrypted secret shares generated by the server;
generating a set of query results responsive to the batch of queries, wherein the set of query results includes the server-encrypted secret shares obtained from the special buckets queried by the batch of queries;
transmitting, to the client device, the set of query results;
receiving, the client device, client-encrypted secret shares, wherein the client-encrypted secret shares are client encrypted versions of the secret shares that were included in the set of query results transmitted to the client device;
transmitting, to the client device, a full set of server-encrypted secret shares, wherein the full set of server-encrypted secret shares includes more server-encrypted secret shares than the set of query results;

receiving, from the client device, a full set of client-server-encrypted secret shares, wherein the full set of client-server-encrypted secret shares are client encrypted versions of the full set of server-encrypted secret shares that were transmitted to the client device;
determining how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device; and
classifying the client device based on how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device.

Clause 9. The system of clause 8, wherein the instructions cause the server device to perform operations further comprising:
removing the server decryption from the full set of client-server-encrypted secret shares received from the client device to obtain a full set of client-encrypted secret shares, wherein determining how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device comprises comparing the client-encrypted secret shares received from the client device to the full set of client-encrypted secret shares obtained by removing the server decryption from the full set of client-server-encrypted secret shares.

Clause 10. The system of clause 9, wherein classifying the client device comprises determining that the client device is malicious based on the comparison indicating that fewer than a required number of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-encrypted secret shares obtained by removing the server decryption from the full set of client-server-encrypted secret shares.

Clause 11. The system of clause 10, wherein the instructions cause the server device to perform operations further comprising:

receiving a set of client-encrypted entity identifiers;
encrypting the set of client-encrypted entity identifiers to create a set of sever-client-encrypted identifiers;
transmitting the set of server-client-encrypted identifiers to the client device.

Clause 12. The system of clause 11, wherein the instructions cause one or more processors to perform operations comprising:
generating a partitioned database in which a database is partitioned into the multiple database shards each having a shard identifier that logically distinguishes each database shard from other database shards, and database entries in each database shard are partitioned into buckets having a bucket identifier that logically distinguishes each bucket in the shard from other buckets in the shard.

Clause 13. The system of clause 12, wherein the instructions cause the one or more processors to perform operations comprising:

adding a special bucket to each shard;
including, in each special bucket, special data that is known to the server device, but not the client device; and
after each query to a given shard, updating the special data in the special bucket of the given shard to maintain privacy of the information contained in the special bucket of the given shard.

Clause 14. The system of clause 13, wherein the client device is configured to perform operations comprising:

generating a set of queries using the set of server-client-encrypted identifiers;
generating a set of decryption keys using the set of server-client-encrypted identifiers;
encrypting the set of queries to create the batch of client-encrypted queries.

Clause 15. A non-transitory computer readable medium storing instructions that, upon execution by one or more data processing apparatus, cause the one or more data processing apparatus to perform operations comprising:
receiving, from a client device, a batch of queries that includes queries to special buckets in each database shard, among multiple database shards, being queried by the batch of queries, wherein the special buckets include server-encrypted secret shares generated by a server;

generating a set of query results responsive to the batch of queries, wherein the set of query results includes the server-encrypted secret shares obtained from the special buckets queried by the batch of queries;
transmitting, to the client device, the set of query results;
receiving, from the client device, client-encrypted secret shares, wherein the client-encrypted secret shares are client encrypted versions of the secret shares that were included in the set of query results transmitted to the client device;
transmitting, to the client device, a full set of server-encrypted secret shares, wherein the full set of server-encrypted secret shares includes more server-encrypted secret shares than the set of query results;
receiving, from the client device, a full set of client-server-encrypted secret shares, wherein the full set of client-server-encrypted secret shares are client encrypted versions of the full set of server-encrypted secret shares that were transmitted to the client device;
determining how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device; and
classifying the client device based on how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device.

Clause 16. The non-transitory computer readable medium of clause 15, wherein the instructions cause the one or more data processing apparatus to perform operations further comprising:
removing the server decryption from the full set of client-server-encrypted secret shares received from the client device to obtain a full set of client-encrypted secret shares, wherein determining how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-server-encrypted secret shares received from the client device comprises comparing the client-encrypted secret shares received from the client device to the full set of client-encrypted secret shares obtained by removing the server decryption from the full set of client-server-encrypted secret shares.

Clause 17. The non-transitory computer readable medium of clause 16, wherein classifying the client device comprises determining that the client device is malicious based on the comparison indicating that fewer than a required number of the secret shares are included in both of the client-encrypted secret shares received from the client device and the full set of client-encrypted secret shares obtained by removing the server decryption from the full set of client-server-encrypted secret shares.

Clause 18. The non-transitory computer readable medium of clause 17, wherein the instructions cause the one or more data processing apparatus to perform operations further comprising:

receiving a set of client-encrypted entity identifiers;
encrypting the set of client-encrypted entity identifiers to create a set of sever-client-encrypted identifiers;
transmitting the set of server-client-encrypted identifiers to the client device.

Clause 19. The non-transitory computer readable medium of clause 18, wherein the instructions cause one or more data processing apparatus to perform operations comprising:
generating a partitioned database in which a database is partitioned into the multiple database shards each having a shard identifier that logically distinguishes each database shard from other database shards, and database entries in each database shard are partitioned into buckets having a bucket identifier that logically distinguishes each bucket in the shard from other buckets in the shard.

Clause 20. The non-transitory computer readable medium of clause 19, wherein the instructions cause the one or more data processing apparatus to perform operations comprising:

adding a special bucket to each shard;
including, in each special bucket, special data that is known to the server device, but not the client device; and
after each query to a given shard, updating the special data in the special bucket of the given shard to maintain privacy of the information contained in the special bucket of the given shard.

Claims

1. A method, comprising:

receiving, at a server device and from a client device, a batch of queries that includes queries to special buckets across database shards;
generating, by the server device, a set of query results that includes server-encrypted secret shares from the special buckets queried by the batch of queries;
obtaining, at the server device and from the client device, client-encrypted secret shares, wherein the client-encrypted secret shares are client encrypted versions of the secret shares that were included in the set of query results;
receiving, at the server device and from the client device, a set of client-server-encrypted secret shares, wherein the set of client-server-encrypted secret shares are client encrypted versions of a set of server-encrypted secret shares that (i) were transmitted to the client device and (ii) includes server-encrypted secret shares that were not included in the set of query results;
determining, by the server device, how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the set of client-server-encrypted secret shares received from the client device; and
classifying, by the server device, the client device based on how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the set of client-server-encrypted secret shares received from the client device.

2. The method of claim 1, further comprising:
removing, by the server device, server decryption from the set of client-server-encrypted secret shares received from the client device to obtain a full set of client-encrypted secret shares, wherein determining how many of the secret shares are included in both of the client-encrypted secret shares received from the client device and the set of client-server-encrypted secret shares received from the client device comprises comparing the client-encrypted secret shares received from the client device to the set of client-encrypted secret shares obtained by removing the server decryption from the set of client-server-encrypted secret shares.

3. The method of claim 2, wherein classifying the client device comprises determining that the client device is malicious based on a determination that fewer than a required number of the secret shares are included in both of the client-encrypted secret shares received from the client device and the set of client-encrypted secret shares obtained by removing the server decryption from the set of client-server-encrypted secret shares.

4. The method of claim 3, further comprising:

receiving, from the client device, a set of client-encrypted entity identifiers;
encrypting, by the server device, the set of client-encrypted entity identifiers to create a set of sever-client-encrypted identifiers;
transmitting, by the server device, the set of server-client-encrypted identifiers to the client device.

5. The method of claim 4, further comprising:
generating a partitioned database in which a database is partitioned into multiple database shards each having a shard identifier that logically distinguishes each database shard from other database shards, and database entries in each database shard are partitioned into buckets having a bucket identifier that logically distinguishes each bucket in the shard from other buckets in the shard.

6. The method of claim 5, further comprising:

adding a special bucket to each database shard;
including, in each special bucket, special data that is known to the server device, but not the client device; and
after each query to a given shard, updating the special data in the special bucket of the given shard to maintain privacy of information contained in the special bucket of the given shard.

7. The method of claim 6, further comprising:

generating, by the client device, a set of queries using the set of server-client-encrypted identifiers;
generating, by the client device, a set of decryption keys using the set of server-client-encrypted identifiers;
encrypting, by the client device, the set of queries to create the batch of client-encrypted queries.

8. A system comprising:

a database configured to store data; and
a server device configured to process queries using the database and execute instructions that cause the server device to perform the method recited by any of claims 1-7.

9. A computer readable medium storing instructions that, upon execution by one or more data processing apparatus, cause the one or more data processing apparatus to perform the method recited by any of claims 1-7.

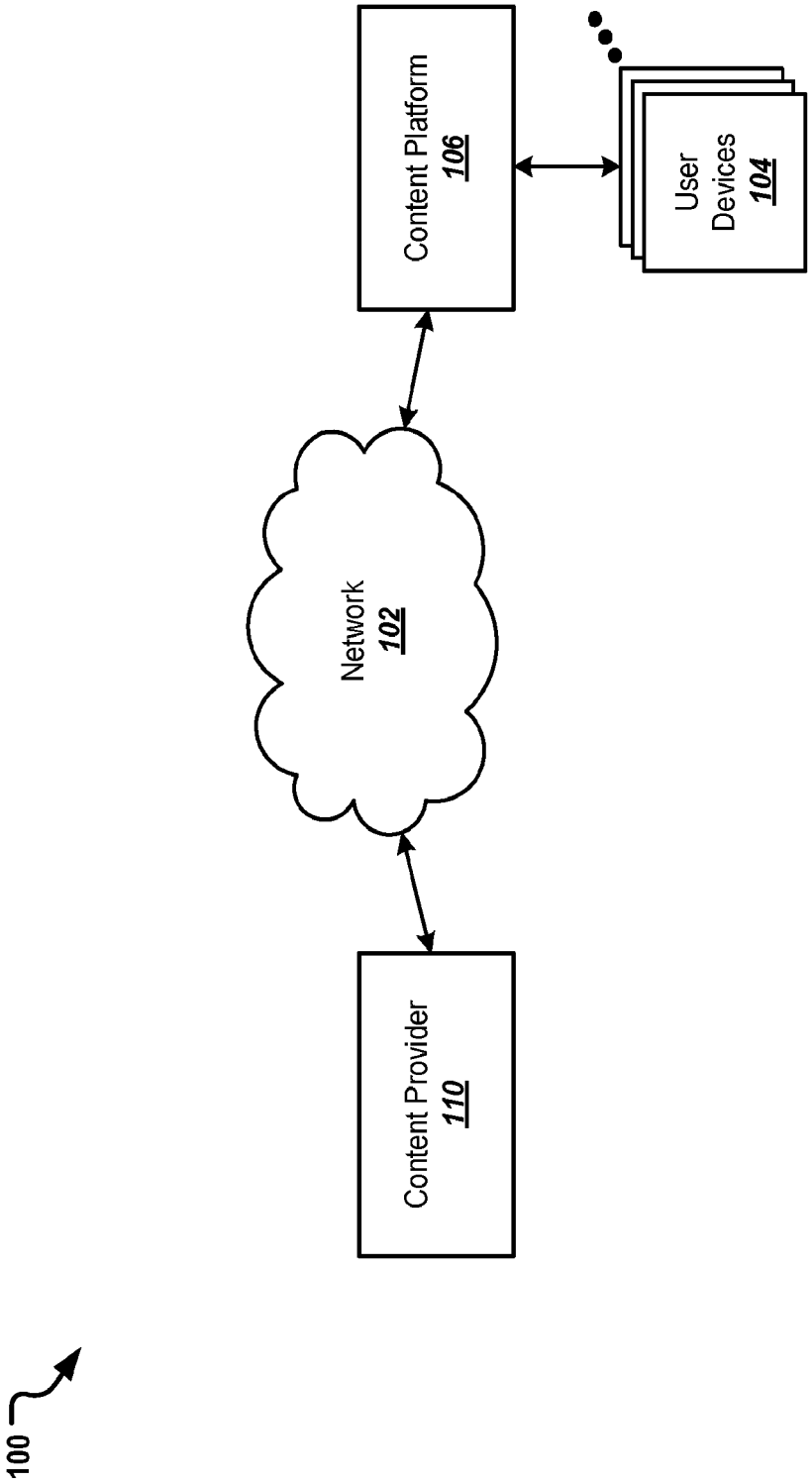
100
Content Provider
110
Network
102
Content Platform
106
User
Devices
104

FIG. 1

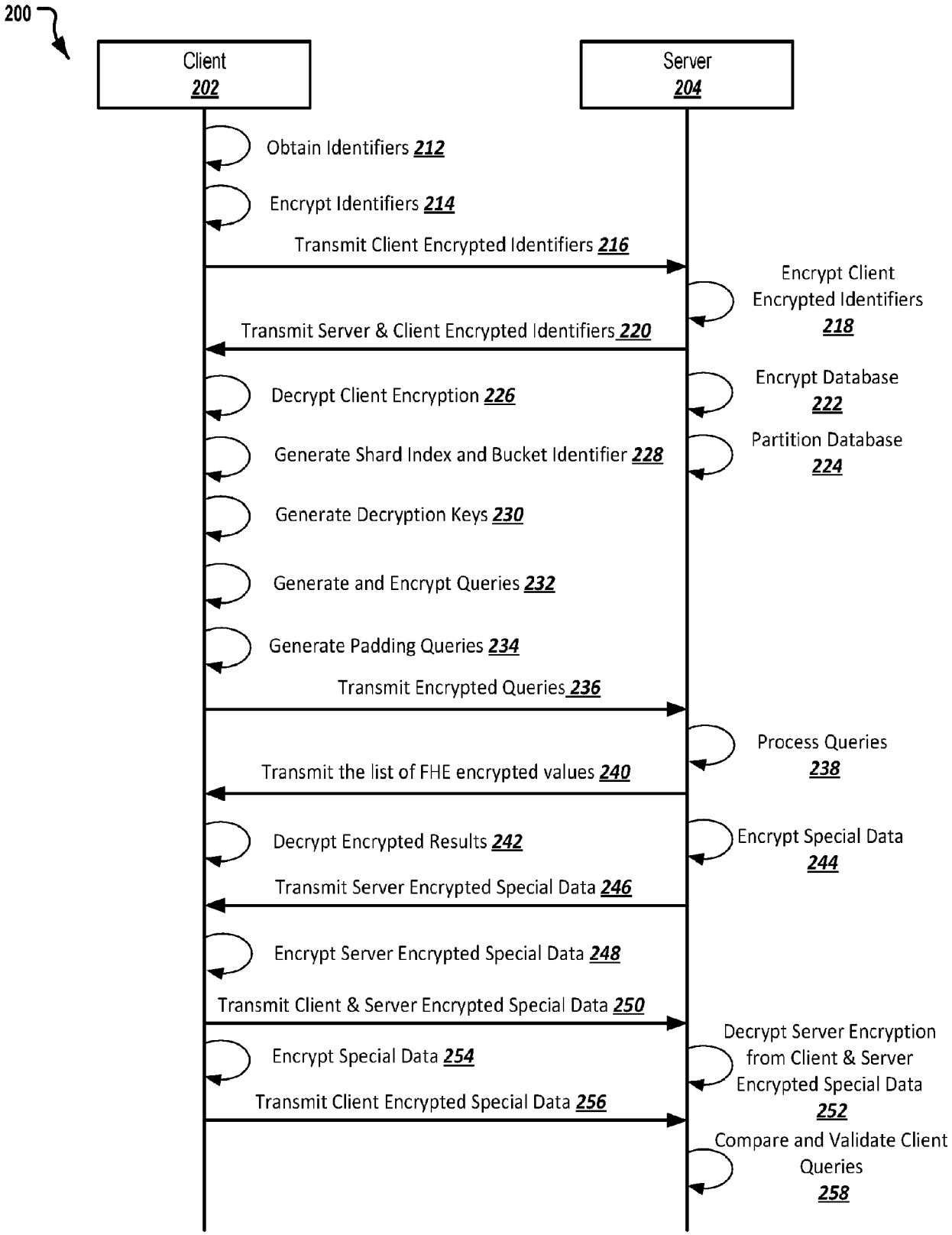
200
Client
202
Server
204
Obtain Identifiers 212
Encrypt Identifiers 214
Transmit Client Encrypted Identifiers 216
Encrypt Client Encrypted Identifiers 218
Transmit Server & Client Encrypted Identifiers 220
Encrypt Database 222
Decrypt Client Encryption 226
Partition Database 224
Generate Shard Index and Bucket Identifier 228
Generate Decryption Keys 230
Generate and Encrypt Queries 232
Generate Padding Queries 234
Transmit Encrypted Queries 236
Process Queries 238
Transmit the list of FHE encrypted values 240
Decrypt Encrypted Results 242
Encrypt Special Data 244
Transmit Server Encrypted Special Data 246
Encrypt Server Encrypted Special Data 248
Transmit Client & Server Encrypted Special Data 250
Decrypt Server Encryption from Client & Server Encrypted Special Data 252
Encrypt Special Data 254
Transmit Client Encrypted Special Data 256
Compare and Validate Client Queries 258

FIG. 2

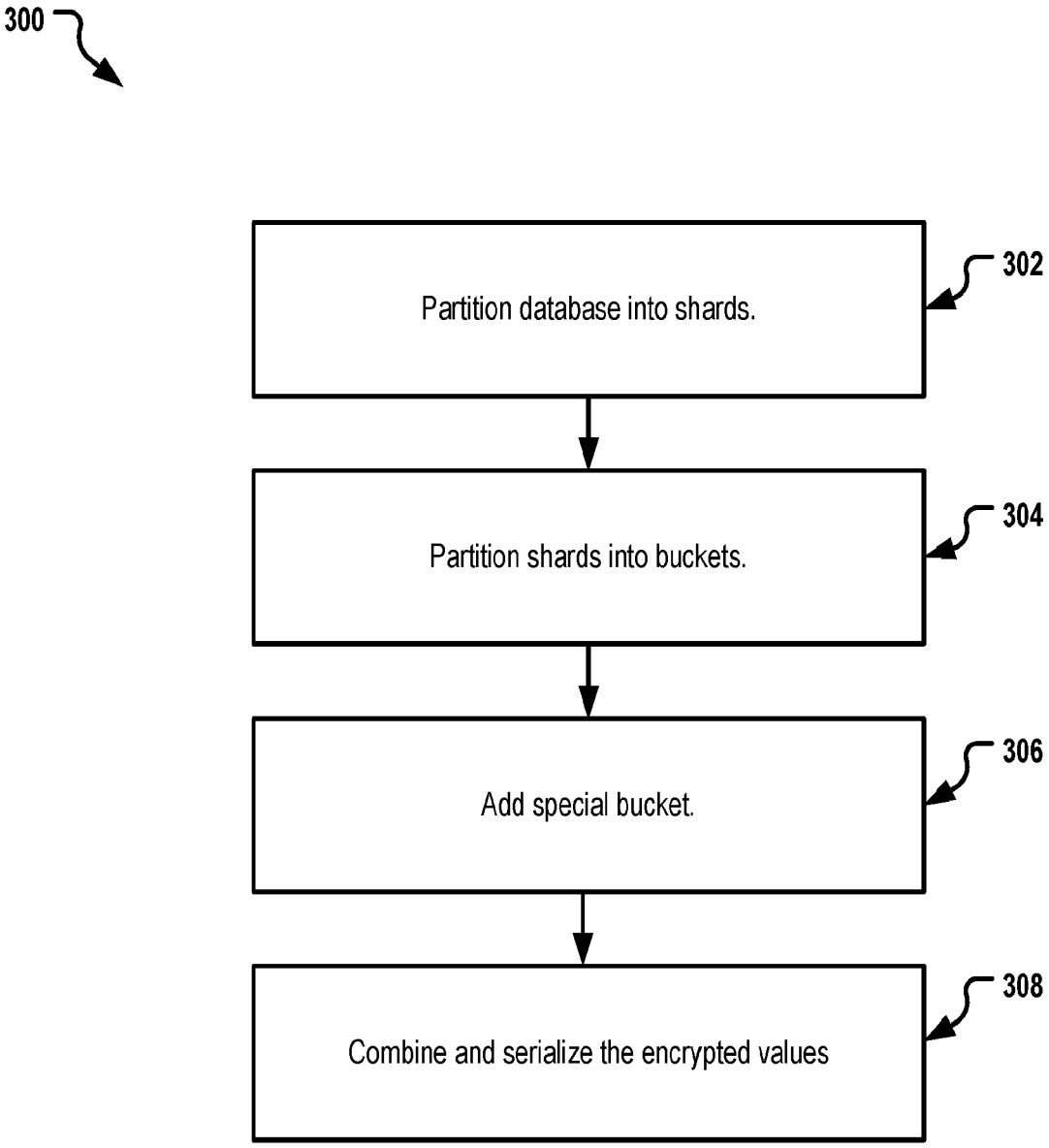
300
Partition database into shards.
302
Partition shards into buckets.
304
Add special bucket.
306
Combine and serialize the encrypted values
308

FIG. 3

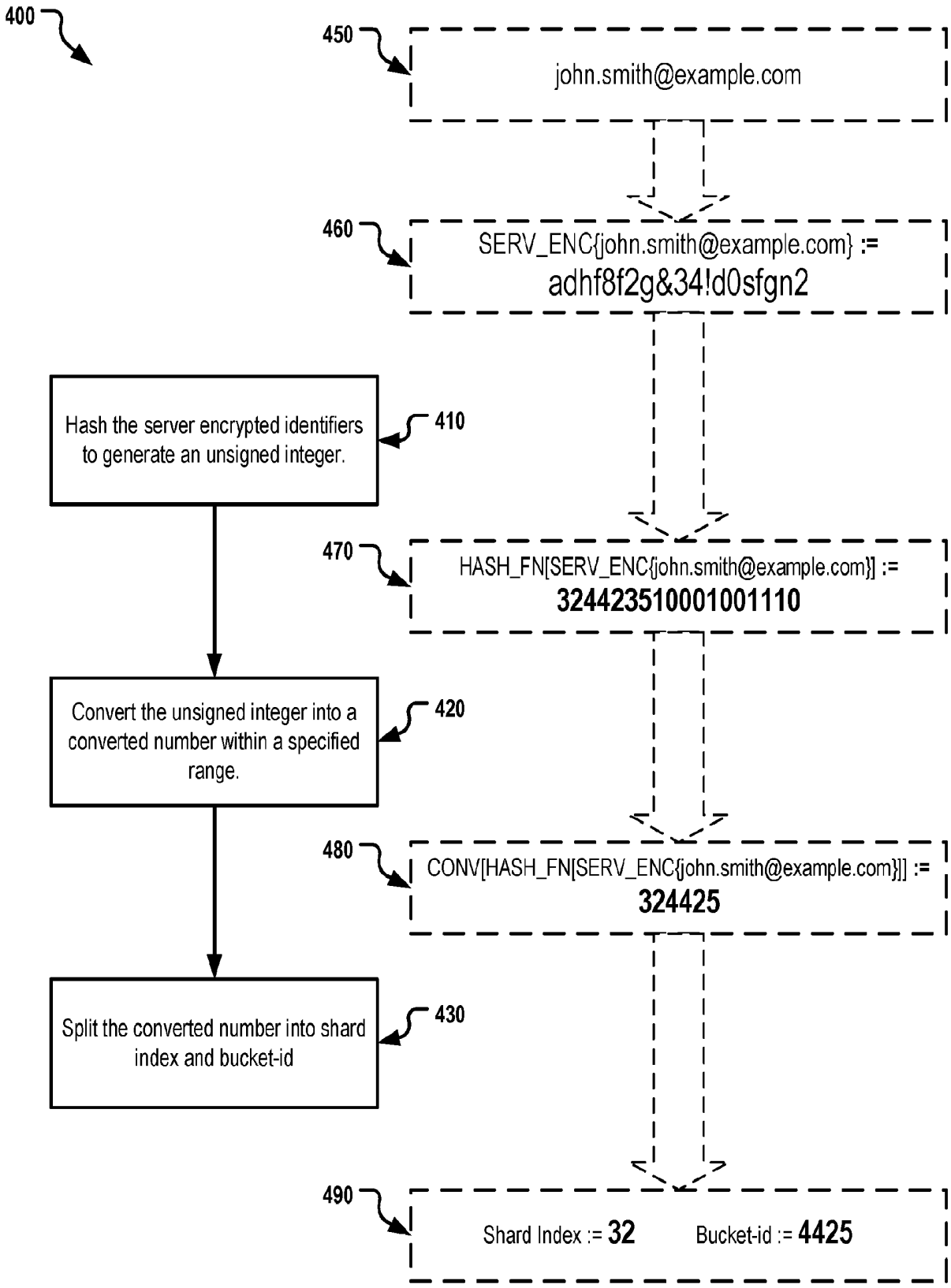
400
450
john.smith@example.com
460
SERV_ENC{john.smith@example.com} :=
adhf8f2g&34!d0sfgn2
Hash the server encrypted identifiers to generate an unsigned integer.
410
470
HASH_FN[SERV_ENC{john.smith@example.com}] :=
3244235100010011 10
Convert the unsigned integer into a converted number within a specified range.
420
480
CONV[HASH_FN[SERV_ENC{john.smith@example.com}]] :=
324425
Split the converted number into shard index and bucket-id
430
490
Shard Index := 32
Bucket-id := 4425

FIG. 4

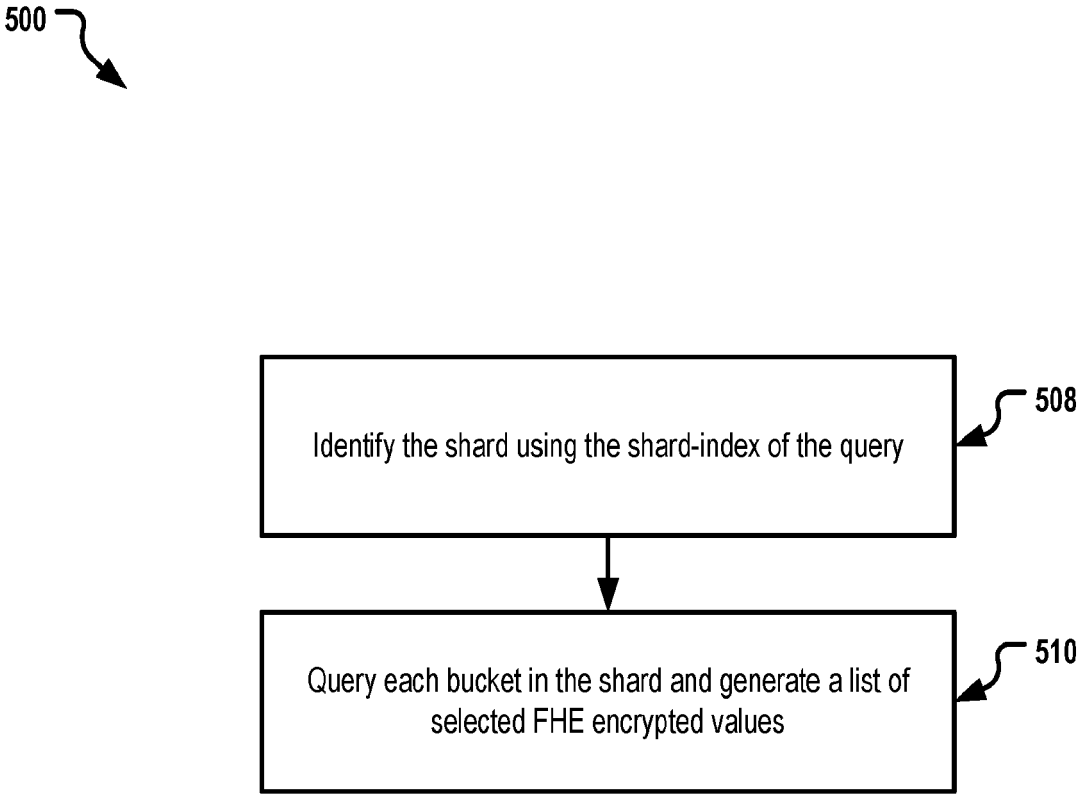
500
Identify the shard using the shard-index of the query
508
Query each bucket in the shard and generate a list of selected FHE encrypted values
510

FIG. 5

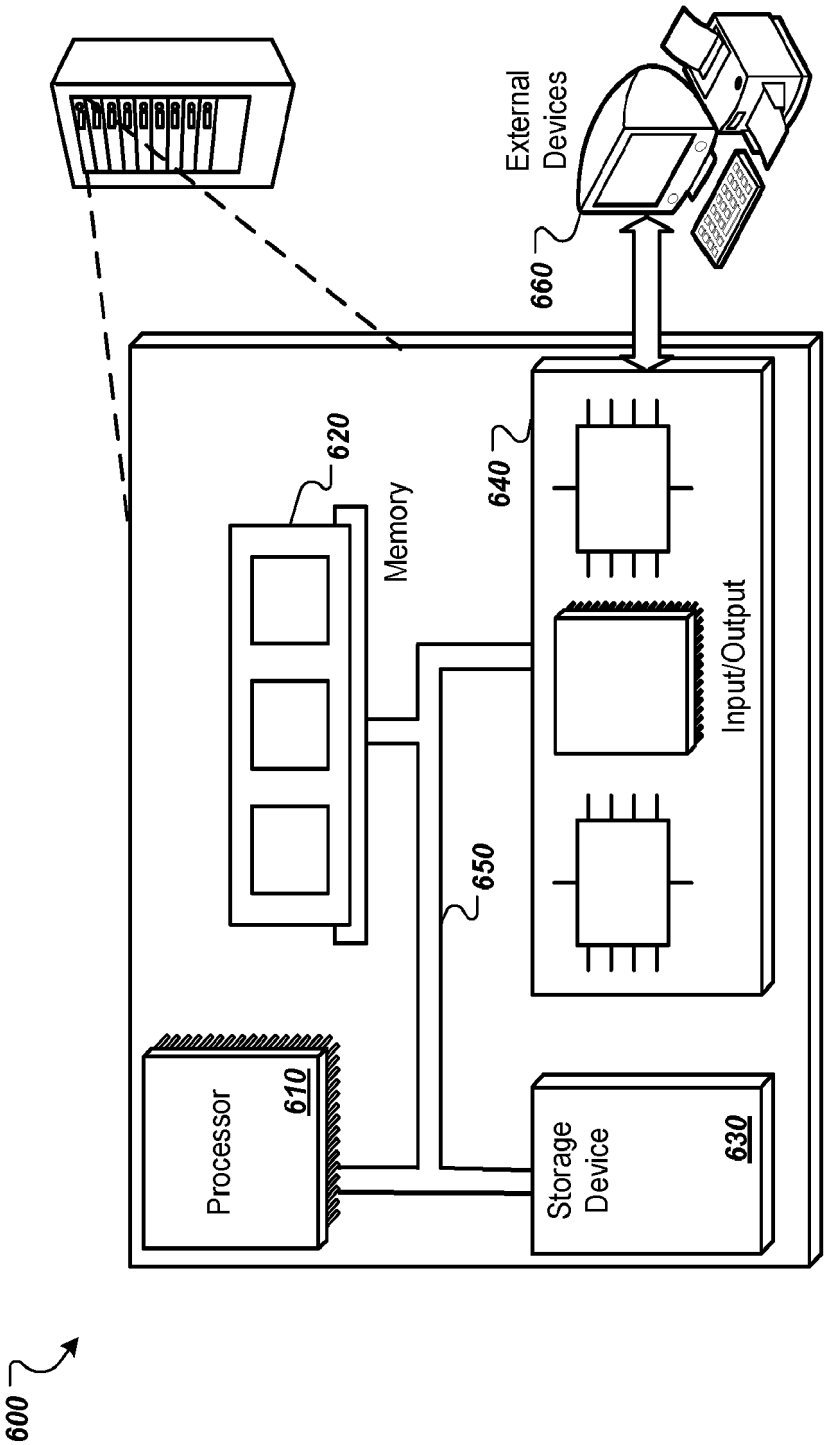
External Devices
660
620
Memory
640
Input/Output
650
Processor
610
Storage Device
630
600

FIG. 6

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- US 63218120 **[0001]**